Increased Server Availability and Flexibility through Failover Capability

Michael R. Barber – Michigan Technological University

ABSTRACT

As computing systems become increasingly mission-critical, a high level of service availability is essential. In order to maintain service availability, it is desirable to have the ability to migrate services from one server to another while having this change remain transparent to the client machines. Although commercial solutions exist which provide automatic failover capability, they are often costly and restrictive.

Manual failover capability is useful for providing service availability in situations where there is a hardware failure, or where a server must be taken down for extended maintenance. The discussion in this document will explore what primitives can be used to construct a failover-capable system, the issues involved in any service migration, and some of the specific details about doing migration of services such as NFS, sendmail, and World Wide Web.

Although implementation-specific examples provided assume a Sun Solaris 2.x operating environment, the use of a logical volume manager, and ethernet connectivity, other flavors of UNIX may also contain the necessary building blocks needed to build a failover-capable system.

Motivation

The systems group, of which I am a member, within the Information Technology – Distributed Computing Systems department at Michigan Technological University determined it was time to upgrade our primary campus servers. One item on the list of desired functionality for the new servers was the ability to do on-the-fly service migration. After examining a few commercial solutions, we decided none of them were appropriate for us.

Due to the nature of a few of our services, it was not an option to increase availability by simply replicating services in more than one place, and using using round-robin DNS, lbnamed [1], or commercial solutions such as Uniq Software Services UPFS [2].

After some experimentation, I found we could get most of what we wanted with a homegrown increased availability solution. To my surprise, the problem of doing manual failover was not as complex as it seemed: a two way failover-capable system can be built using two computers, one or more multi-ported disks, and three or more network interfaces on each host.

Foundation

An important concept to understand is that, in this model, a group of services is tied to a hostname. This hostname is considered "public" meaning it is what clients use to acquire a given service. The unusual detail is this public hostname is not permanently tied to a particular host, and may be thought of as a pointer to the server which is currently providing the given service set.

The two main pieces of the puzzle needed to build a failover-capable system are the ability to easily swap file system ownership between two hosts, and a network configuration which allows both hosts to impersonate each other. The ability to easily move file system ownership between two hosts is best accomplished by hardware such as multi-ported disk arrays (e.g., Sun's Sparc Storage Array) which allow more than one host to be connected to the array simultaneously. Although it is possible for two hosts to share a single SCSI chain, using a multi-ported disk array is a much cleaner solution.

The two logical volume managers Sun offers (Solstice Disk Suite and Veritas Volume Manager) provide support for multi-host disk arrays. The implementation-specific examples in this paper use the multi-host features of Veritas Volume Manager, although Solstice Disk Suite has a feature called "disksets" which can be used in a similar manner. While logical volume management software is not absolutely required in order to implement a failover configuration, it greatly simplifies the problem, and also provides features which can be used to provide a higher level of data availability, such as disk mirroring or RAID-5.

When configuring the two machines it is important they mirror each other as closely as possible. Each file system on the multi-ported disk array which may need to move from host to host must look the same to both hosts. If both hosts are not configured almost identically, problems may arise at a later date when a service migration becomes necessary.

Network Connectivity

Three network interfaces are required on each host to do two-way failover in which both machines are servers and either machine is capable of filling in for the other machine. The concepts in this paper can be scaled to N nodes, given a N-ported disk array and M+1 network interfaces on each node; where $N \geq 2$ and M is the number of service sets any one server will ever need to run concurrently. Situations where $M > N$ require extra complexity. This paper will focus on bipolar two-way failover (M=N=2).

So as not to confuse switches and other hosts on the network which maintain an arp cache that maps ethernet address to IP address, each host in a failover configuration must be able to impersonate the other host on both a ethernet and IP level. Furthermore, it is important to have a network topology that will allow this.

If your pair of machines are behind a smart switch that maintains a bridge table, the switch must allow for the possibility of an ethernet/IP pair to move from one port to another and update its bridge table accordingly. From the switch's perspective, a service migration should look as though the server was unplugged from one port and plugged it into another.

With the smart hub in use at our site, having the hub update its tables was as simple as running a program on the machine to which services were migrated, which generates traffic on its NIC (Network Interface Card, in this case the ethernet interface) so the switch would notice the change. To accomplish this, I used a simple program which does a UDP broadcast which sends the message "WAKE UP" out each interface to the "discard" port of all other machines in the same broadcast domain.

Each host in a two-way failover configuration will have one private and two public interfaces. Since each of the public interfaces on one node will have a counterpart on the other node, there will be only two unique ethernet addresses among the four public interfaces on two machines. Each private interface will have its own unique ethernet address, bringing the total on both machines to 6 network interfaces with 4 unique ethernet addresses, and at least 4 unique IP addresses.

Virtual network interfaces can often be substituted for physical interfaces; unfortunately this cannot be done for the three network interfaces required for two-way failover. Although this may change in a future version of Solaris, as of Solaris 2.6, a virtual interface must assume the same MAC address as the physical interface which it is associated to. However, virtual network interfaces may still be associated with public interfaces and used for virtual mail domains and virtual web servers.

Sun workstations obtain a default MAC address from NVRAM, instead of from the ethernet hardware [3]. Because of this, any add-on ethernet interfaces will by default assume the same ethernet address as the on-board ethernet interface. This behavior can be overridden by setting the option "local-mac-address?" to "true" from the prom monitor, which will cause each NIC to have a unique MAC address. In a failover system with three ethernet addresses, you will need to explicitly redefine two of the addresses, therefore setting this option in the prom is not mandatory.

You will need to create two MAC addresses, each of which will be assigned to one NIC on each machine. Although Solaris will let you assign nearly any ethernet address to the interfaces, you should pick an address which is IEEE 802.3 [4] conformant.

A standard 48 bit ethernet address is comprised of six colon-separated octets (8 bit groups). The first (left most) octet should have the first (right most) bit unset and the second bit set. The first bit indicates individual (0) or group (1) destination address. The second bit indicates a global (0) or local (1) address.

HOST A	hme0	hme1	hme2
hostname:	campus0	server0	server1
ethernet address:	8:0:20:81:df:f4	2:0:8d:db:46:1	2:0:8d:db:46:b
metric:	1	0	0
normal state:	up	up	down

HOST B	hme0	hme1	hme2
hostname:	campus1	server0	server1
ethernet address:	8:0:20:85:84:45	2:0:8d:db:46:1	2:0:8d:db:46:b
metric:	1	0	0
normal state:	up	down	up

Table 1: Sample interface configurations.

Setting the second bit to indicate "local" means the address was not assigned by IEEE. This guarantees it will not conflict with the addresses assigned by vendors, but there is no assurance it is not in use elsewhere [4,5].

When creating local ethernet addresses, many sites set the second bit in the first octet as per IEEE 802.3 specifications, and set the last four octets to the hexadecimal equivalent of their IP address. This scheme gives the ethernet address the form of 2:0:w:x:y:z (w, x, y, z in hexadecimal) where the machine's IP address is w.x.y.z (normally given in decimal). When used consistently, this scheme ensures no unwanted duplication of addresses.

You can use the ifconfig command to set the MAC address on a NIC, for example: "ifconfig hme1 ether 2:0:20:81:d2:c" will set the ethernet address of hme1 to 2:0:20:81:d2:c.

Table 1 is an example of how one might configure the three interfaces on each host.

The interface hme1 has the same hostname and same ethernet address associated with it on both machines. However, both interfaces will never be up at the same time. The interface hme2 is configured in a similar manner. In the above case, the clients will use the names server0 and server1 to obtain services.

From a network perspective, whichever machine has interface hme1 up is server0 and whichever machine has interface hme2 up is server1. See Figure 1 for a diagram of the system layout during non-combined operation.

As shown in Figure 2, after a service migration is completed, one host providing both service sets will have both public interfaces hme1 and hme2 up, while the other host will have both public interfaces down.

A third "private" interface is needed so the machine which has both public interfaces down may still have network connectivity. In the above case, hme0 with names campus0 and campus1 are the private interfaces of the two servers.

A "metric" may be associated with a network interface. This number which is normally zero indicates how may "hops" should be added to an interface when choosing the best route. The private interface must be configured with a higher metric, giving it a less desirable route so the public interfaces will be used by default. For example, "ifconfig hme0 metric 1" will to increase the metric of hme0 from its default zero to one.

If an interface is assigned a metric of one when the system is booted, and at a later time another interface is brought up with a metric of zero as in the case of a failover configuration, the private interface must

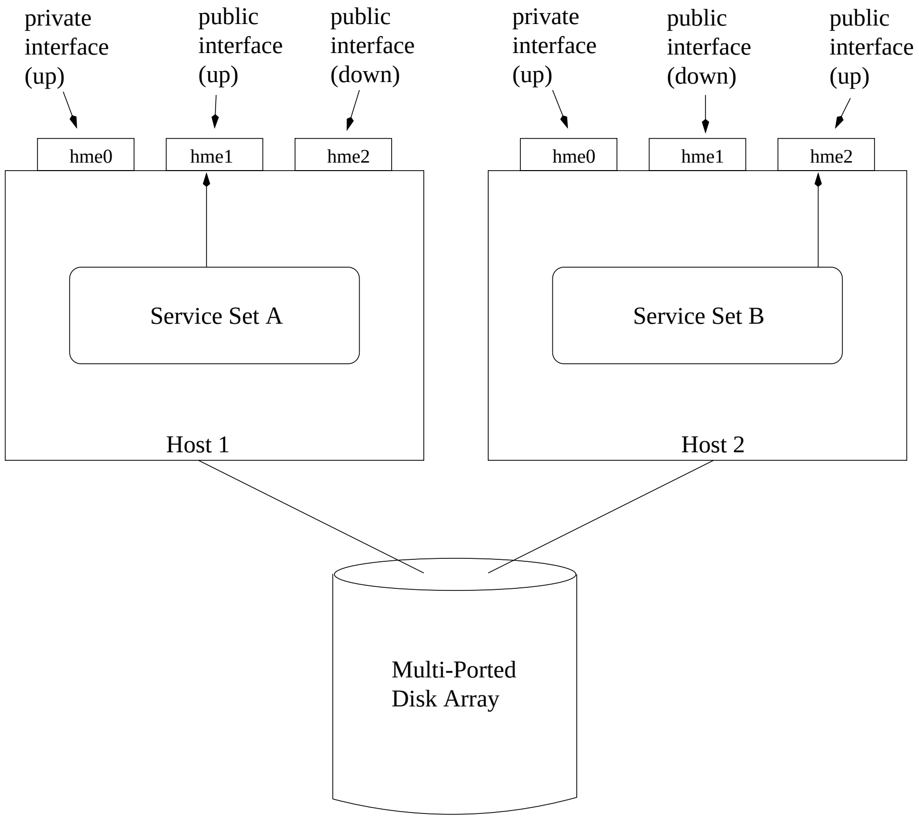

Figure 1: Normal bipolar operating mode.

be taken down and brought back up. Perhaps this will change in a future release of Solaris or as the result of an operating system patch, but Solaris 2.5.1 does not seem to update its routing tables when a second NIC comes up which is connected to the same subnet as the first interface, even if it has a lower metric.

Given a Solaris 2.5.1 or earlier operating environment it is nondeterministic which NIC will be used when multiple interfaces with the same metric are connected to the same subnet. This is generally not a problem when running in bipolar mode since there are only two interfaces up and one was explicitly given preference by setting the metric on the other interface to be higher. However, when operating in combined mode there are two interfaces connected to the same subnet with the same metric. This may cause minor problems for a few services.

Unpredictable choice of interface in this situation should be less of an issue in Solaris 2.6, which provides ‘‘interface groups.’’ The Solaris 2.6 kernel maintains a table of IP addresses on which each client has contacted the server. This data is used to determine which IP address to send the reply from [6]. Since clients use the public interfaces to obtain service, the correct interface should be chosen for communication back to the client. The use of interface groups in Solaris 2.6 can be enabled by setting parameter ‘‘ip_enable_group_ifs’’ in the /dev/ip driver to ‘‘1’’ with ndd.

Service Failover

Generic Method

With most services, such as simple web or directory service, doing failover is straightforward. When these services are migrated, it is an easy matter of stopping the service on one host and starting it on the other. If care is taken when performing the migration of services, impact on the clients can be minimized.

The basic algorithm for migrating a service set from host A to B is as follows:

1. Check to see if Host A is still providing the service to to be migrated.
2. If Host A is still providing service set, shell to Host A and:
   a. Stop services which need to be migrated.
   b. Unmount disks which need to be migrated.
   c. Release any locks which may prevent the other host from acquiring the disks to be migrated.
   d. Bring down public NIC associated with service set.
3. Import and mount disks associated with service set being migrated. This should include a check

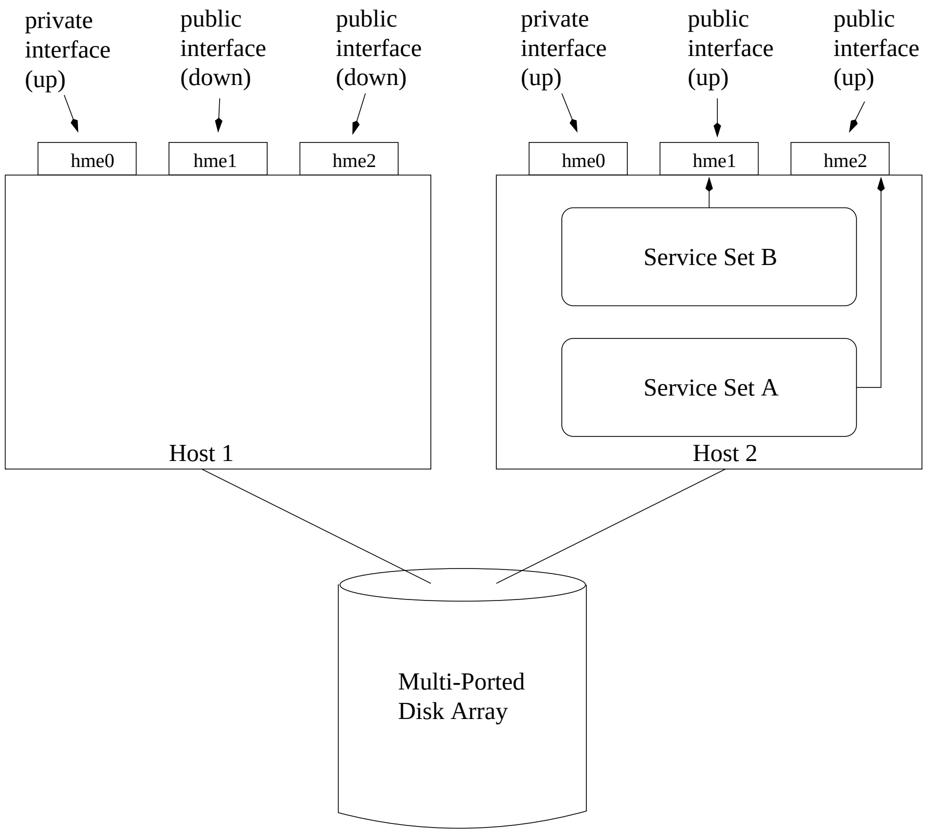

Figure 2: Combined operating mode.

to see if the file system needs a consistency check (i.e., fsck).

4. Bring up public NIC associated with service set.
5. Start services which host A was previously providing.

In our environment, we were able to configure all of our production services in such a way that service migration is possible. However, some services are more difficult to provide within this framework, and others simply cannot be provided in this manner. The following sections describe some of the implementation-specific details for services which are not as straightforward to migrate.

NFS

NFS file handles contain all the information a NFS server needs to distinguish an individual file [7, 8]. In Sun's NFS implementation, this information includes the major and minor device numbers of the file system being nfs exported. Therefore, it is essential the device major and minor numbers of the file system to be migrated be identical on both hosts. If the major/minor device numbers do not match, when NFS service is migrated, the clients will have stale NFS handles.

Either of the two logical volume managers mentioned above will take care of the device names so they are the same on both hosts; however the major device numbers will not necessarily be the same. The relevant entry in /etc/name_to_major for the Veritas Volume Manager 2.3 is "vxio," and for Solstice Disk Suite it is "md." The number associated with this entry should be the same on both hosts.

If the entry for "vxio" or "md" in the /etc/name_to_major file is the same on both hosts, the device files will have the same major and minor device numbers. However, if these numbers are not the same, you will need to edit the /etc/name_to_major file and set one of the numbers listed for "vxio" to be the same as on the other host. Be sure to look through the file to make sure the number you are setting one of the "vxio" entries to is not already in use. After changing this entry in /etc/name_to_major you will need to do a reconfiguration boot. (i.e., boot -r)

Doing NFS service migration with a writable file system is somewhat risky, but in most cases is an acceptable risk given the nature of the situations when failover is needed. When NFS service is migrated from one server to another, any kernel locks (fcntl, flock, etc.) will be lost, while "dot" locks will be preserved. From the perspective of the NFS client, it will look as though the NFS server was rebooted. However, unlike after reboot, the NFS server will not have a list of which hosts which held kernel locks and need to be contacted. It may be possible to preserve kernel locks by migrating the contents of /var/statmon thereby making the NFS server request kernel lock information from the clients as if it had be rebooted.

NFS over UDP handles failover quite well. Assuming there is not a big time gap between when server A stops serving and server B starts serving, the NFS clients do not even notice when service migration has occurred.

NFS over TCP has a few quirks. If NFS services are migrated from server A to server B, the clients "hiccup," complaining "NFS server . . . not responding" and then immediately "NFS server . . . ok." This is caused when NFS clients try to communicate to the failover server which the client believes to have an open TCP stream. The NFS server returns a TCP RST to the NFS client indicating there is no open TCP stream, which causes the NFS client to "hiccup." Upon receipt of the TCP RST the NFS client then sends the appropriate TCP SYN packets to establish a new TCP connection.

A problem occurs with NFS over TCP if you then migrate NFS service from server B back to server A, without either rebooting server A or waiting approximately thirteen minutes between service migration. Even after NFS services are migrated away from a machine and the public NIC is taken down and unplumbed, nfsd maintains an open TCP stream to the nonexistent interface. After thirteen minutes of trying to use the no longer functional interface, nfsd will destroy that TCP stream. If NFS services are migrated back before the nfsd has closed this stream, the NFS clients will hang because of mismatched TCP sequence numbers. This should not be a problem, since in most circumstances where service migration is needed, the duration will be longer than thirteen minutes, or the machine originally hosting the services will be rebooted.

After a migration of NFS services has been performed, one consequence is the data stored in /etc/rmtab is very likely no longer accurate. I have found no negative side effects other than the showmount command giving an incorrect list of hosts.

Another NFS over TCP issue is the case in which your failover-capable NFS server also acts as a NFS client in that it NFS mounts file systems from remote machines. Because the public network interfaces will have a higher preference (a lower metric) than the private interface, the public network interfaces will be used when NFS mounting a remote file system. In the case of a service migration, the public interface being used may be taken down, thereby causing the machine to hang on the apparently downed NFS server. One workaround for this problem might be to temporarily unmount all remote file systems while performing a service migration.

sendmail

Although DNS MX records can be used to provide fallback mail service, this will not provide data availability while the host which spools e-mail is down. Berkeley sendmail 8.8.x can configured to allow for service migration.

In a dual-host configuration it may be desirable to run a sendmail process on each host, but not necessarily configured to perform similar functions – for example, one host may spool mail and the other host may handle off-site delivery. One solution to this problem might be to have three sendmail configuration files: one for host A, one for host B, and another to allow a single sendmail process to fulfill the roles of both machines for operation in a combined post-migration mode.

There is a cleaner solution which allows sendmail to operate on a combined machine using the same configuration files which it uses when running on both hosts. This solution does require a few changes to your sendmail configuration file, a few compile time options, and as of Berkeley sendmail 8.8.7, a small source code modification.

To provide the basis for sendmail service migration, the sendmail binary, configuration file, mqueue, and possibly a mail spool, need to be located on the multi-ported disk array. This may require various compile time options to be set in order to specify an alternate mqueue directory and sendmail configuration file location.

So that sendmail will read its sendmail.cf configuration file and store its sendmail.pid in a nonstandard location, add the flags provided in Listing 1 to "ENVDEF" found in your sendmail makefile. Of course, replace the path names with the ones correct for your system.

Within the sendmail configuration file, you will need to change the default mqueue directory to a path which resides on the multi-ported disk array. To change this, add the following to your m4 configuration file:

```
define('QUEUE_DIR',
  '/mailgate/var/spool/mqueue')
```

For those not using m4 macros, modify the QueueDirectory entry in your sendmail configuration file to reflect the correct path name.

Version 8.7 and beyond of Berkeley sendmail has a configuration option called "DaemonPortOptions." Using this option, you can force sendmail to bind to a specific IP address for daemon mode, which can be used to make sendmail use a specific public NIC associated with the given IP address.

This feature is useful in a dual-host configuration when sendmail services have been combined on one host; specifically, two sendmail processes can be run on one host, both in daemon mode, each binding to a different IP address. This allows sendmail to function the same whether it is one process on each host, or two processes on one host.

To force sendmail to bind to a specific IP address when running in daemon mode, add the following m4 macro configuration file option:

```
define('confDAEMON_OPTIONS',
    'Addr=mailgate')
```

The sendmail configuration file equivalent to this is:

```
O DaemonPortOptions=Addr=mailgate
```

Of course, use the public host name of your mail server instead of "mailgate."

Although sendmail can be easily configured to bind to a specific IP address for accepting incoming SMTP connection in daemon mode, as of Berkeley sendmail 8.8.7 there is no option to force sendmail to bind a specific IP address when establishing remote connections. This has the potential to cause the header "X-Authentication-Warning" to be inserted into e-mail when sendmail connects to a remote host but reports its hostname to be different than the name which maps to the IP address it used for the outgoing connection.

To resolve this outgoing SMTP IP address problem, I found that a one line addition to the sendmail source code can be used to force sendmail to use the same IP address for outgoing connections as listed in the configuration file used for accepting incoming SMTP connections. This sort of functionality is a planned feature for a future release of Berkeley sendmail [9]. This may not be necessary under Solaris 2.6 due to the feature called "interface groups" mentioned above.

In the file daemon.c as obtained from the Berkeley sendmail 8.8.7 source code distribution, find the line:

```
i = connect (s,
   (struct sockaddr *) &addr,
   addrlen);
```

Insert the following right before the connect line you just found:

```
(void) bind(s,
    (struct sockaddr *) &DaemonAddr,
    addrlen);
```

You may want to surround this line by "#ifdef" preprocessor statements for conditional compilation, since this source code modification may not be appropriate for multi-homed hosts on multiple physical networks.

```
ENVDEF=-D_PATH_SENDMAILCF=\"/mailgate/etc/mail/sendmail.cf\" \
      -D_PATH_SENDMAILPID=\"/mailgate/etc/mail/sendmail.pid\"
```

Listing 1: Defining non-standard files.

When running in a combined mode, there is the potential for one sendmail process to try to talk to the other sendmail process. It is desirable for the sendmail processes to communicate as if they were on separate hosts. A modification of the configuration file is necessary so sendmail does not erroneously detect it is talking to itself and disallow it to prevent a mail loop. This can be solved by hard coding macros "$w" and "$j" in the sendmail configuration file to be the unqualified host name and fully qualified hostname (respectively) of the public interface being used. To force correct values into "$w" and "$j", use the following in your m4 configuration file:

```
LOCAL_CONFIG
Dwmailgate
Dj$w.$m
```

If you are not using the m4 macros, just drop the "LOCAL_CONFIG" line and put the "Dw" and "Dj" lines in your sendmail.cf file.

World Wide Web (httpd)

In our environment, doing web service failover was quite simple. Our web site does mostly "vanilla" web file serving with very few cgi scripts.

Most modern web servers have an option which allows you to bind a web server to a specific IP address. For example, the option with the NCSA web server is "BindAddress" found in "httpd.conf." Set this to the IP address or hostname of your public interface through which clients obtain web service.

Because a web server can be bound to a specific interface or IP address, you may run web servers on both hosts in your failover configuration and expect them to operate in a similar manner when running in combined mode.

inetd

Services which are managed by inetd can also be easily migrated from one host to the other. To accomplish this, as many as three separate inetd configuration files may be needed: a generic inetd configuration file listing services provided by both machines, and an individual inetd configuration file for each of the two machines listing only the inetd managed services unique to each machine.

Solaris allows more than one inetd process to run simultaneously, and inetd will accept the name of an alternate inetd.conf file on the command line. Since there is the potential for all three inetd configurations to be running concurrently on one host there may not be any conflicts between the configurations in which more than one service is associated with the same port.

Behind the Scenes

In addition to planning service migration of daemons which provide service to end users, it is important to consider those services which work behind the scenes, such as cron and system backups. The following sections describe how to schedule these events so they happen consistently. If a service migration happens while one of these jobs is running, it may need to be manually restarted after the migration occurs.

cron

It is a relatively simple matter to ensure that cron jobs are executed in the same manner in a combined failover mode as they do when services are spread across both machines. This requires identical crontab entries on both machines where each job is preceded by a test to determine if determine if the given cron job should be run.

For example, if one service set needs a cron job to regenerate a class list daily, a crontab entry might look like:

```
30 0 * * * [ -x
     /opt/class_list/generate ]
     && /opt/class_list/generate
```

The test expression assumes if the program "generate" exists, the service set which requires its use must be running on localhost. The test expression must therefore always test for a file or directory which will only be present when the given service set is running on localhost.

One unfortunate aspect of this is if you have an error in the crontab file, it may escape detection for some time. Errors in the test expression may silently cause a job to not run. If a service set's crontab entry is not correct on both hosts, an error may not cause problems until service migration has occurred.

Backups

Since backups must occur even when services have been migrated, it is important to put thought into the backup software configuration.

In the situation where a remote host performs the backups of both servers, the configuration is simple – just have the software use the public network name of the service set which contains the file systems to be backed up. The backup software should then be able to always find the data to be backed up.

In our environment, both machines have their own local tape drives and run non-commercial backup software. To ensure backups happen consistently, the backup schedules are identical on both machines. This causes the tape indexes to be the same on both tapes, however only a small dump record is written to tape for file systems on the backup schedule which are not present on the host doing the backups.

This backup strategy works well for full level zero dumps, but it may cause problems for incremental dumps. Although both machines will have identical backup schedules, the time stamps listed in /etc/dumpdates for both hosts will be close, but not the same. Therefore, there is the potential to miss data when performing an incremental backup of a file system which was resident on the other host at the time of the last backup. One possible solution to this would be to write a program to keep /etc/dumpdates files synchronized, and populated with correct time-stamps.

System Reboot

When a host provides a service set, whether it normally resides there or has been migrated from another host, it is important services be resumed if appropriate after a reboot. This issue can be addressed in a boot-up rc script using the following strategy:

For each service set which I may provide,

1. check to see if another host is providing given service set; if so skip item 2 on this list.
2. If the given service set was being provided at the time of system shutdown, start running that service set.

Performing this check on boot-up will ensure that if a machine goes down while hosting a given service set, it will restart the service set after boot-up provided there is no other machine currently offering said service set. If using Veritas Volume Manager, it is simple to determine if a service set was up on a machine by checking for the existence of the directory /dev/vx/dsk/<set_name>.

An Example Implementation

This section contains details about how I chose to implement the failover system in use at Michigan Technological University. It is certainly not a definitive guide on how a failover system must be configured. Use it in the context it is intended – as an example.

The software I wrote for service migration is actually just a small collection of Bourne shell scripts and a short C program. The scripts are written to be simple, effective, and easily configurable.

- acquire – used to migrate given set of services to localhost.
- release – helper script for acquire usually run via rsh instructing the machine providing a service set to stop those services and release the relevant file systems.
- setup_interface – helper script for both acquire and release which is used to configure, bring up, and down the ethernet network interfaces associated with given service set.
- notify_switch – C program required after a service migration if both servers are on separate switched ports on a hub. This simple C program does a broadcast which generates traffic on all its network interfaces and causes the hub to realize the IP/MAC address pair moved to a different port.
- unlockdg – forcibly remove a host lock on a given disk group. This script is useful in the case of an ungraceful service migration where the host which held the lock is down or unable to release the lock.

One way I attempted to keep these scripts simple was to use the name of the public interface to also label the Veritas Volume Manager disk group. Therefore, there is a direct mapping between a set of services, the public host name, and the disk group name. In order to put as little configuration information in the scripts as possible I created a volume called "config" on each disk group which contains configuration information about that particular service set.

The "config" volume contains a small collection of binaries, configuration files, and rc startup and shutdown scripts. This file system gets mounted as /<service_set_name>. The files in "config" are in a similar arrangement to where the normal files are, except with "/" relative to /<set_name>. You might find directories such as /etc, /etc/mail, /etc/dfs, /var/spool/mqueue, /usr/lib, and /sbin containing configuration information essential for the given service set, such as vfstab, dfstab, sendmail.cf, and inetd.conf.

The "acquire" script examines the contents of this configuration file system to determine which file systems to mount, NFS export, which services to start via rc scripts, and if any services require an inetd process to be run for that service set. The release script uses the rc scripts on this file system to stop services which were started via rc scripts.

Since using the rc script to stop services from a given service set may not always kill every process using the file system which needs to be unmounted, the "release" script makes use of the "fuser" command to list processes using a given file system. Processes are first sent a polite SIGTERM, and if still running after one second, they are sent a SIGKILL. If there are any kernel locks on a file system to be unmounted, the lock manager must be stopped before the unmount and started after the unmount.

Increased Availability or High Availability?

While the application of the techniques presented within this paper may give extra flexibility and allow uptime when there would normally be service interruption, this is in no way a HA (High Availability) solution. Operator error, poor fault detection, and failure of the network fabric are just a few items which may cause service interruption. Sites which require a guaranteed percentage of uptime should explore commercial HA solutions which will perform fault detection and automatic failover.

Why stop with manual failover? On the surface it may not seem difficult to write a program that would perform the functions discussed in this paper in an automatic failover capacity. Unfortunately, the problem of doing failover gets considerably more complicated when you intend for the system to run on autopilot and migrate services without any human intervention.

The main problem with automatic failover is one node detecting when the other node is down and determining if it is a situation in which it should attempt to run the downed node's services. Incorrect detection of failure may lead to inappropriate failovers, which have the potential to bring more instability to your system than homegrown automatic failover can hope to provide.

Availability

Example source code for performing service migration may be obtained from http://www.it.mtu.edu/failover/ .

Conclusion

It is possible to build a framework through which service availability can be increased by separating the service from the server. In the event of a server hardware failure or the need for system maintenance, the system administrator can easily migrate the services to a different machine. In addition to increasing service availability, this may also reduce the stress put on a system administrator who, for example, just had a CPU fail in one of the primary production servers.

Of the high availability solutions prevalent today, some provide attractive functionality, but there is still much to be done in this area. As node clustering and single system image become a mature high availability alternative, many of the topics which this paper explores may become non-issues.

Acknowledgments

Thanks to Thomas Dwyer III of Sun Microsystems for his technical guidance with this project and allowing me to use him as a sounding board for my ideas. His insight and suggestions have proven invaluable.

Thanks to my coworkers and my supervisor, Ann West, for supporting this project. Also thanks to the folks who proofread and critiqued my paper – they have certainly helped make this paper more readable.

Author Information

Michael R. Barber is currently a Senior Systems Programmer in Michigan Technological University's Information Technology department, and holds a bachelors degree in Computer Science. You can reach Michael electronically at barber@mtu.edu.

Bibliography

[1] Schemers, Ronald J., "lbnamed: A Load Balancing Name Server in Perl," *Proceedings of the USENIX Systems Administration (LISA IX) Conference*, pp. 1-11, Monterey, CA, September, 1995.

[2] Uniq Software Services, "UPFS – A Highly Available Filesystem," http://www.uniq.com.au/products/upfs/UPFS-WhitePaper/UPFS-WhitePaper-1.html .

[3] "Frequently Asked Questions about Sun NVRAM/hostid," http://www.squirrel.com/squirrel/sun-nvram-hostid.faq.html .

[4] ANSI/IEEE Std 802.3, *Carrier sense multiple access with collision detection (CSMA/CD) access method and physical layer specifications*, 1996 Edition, pp. 12-14.

[5] Andrew S. Tanenbaum, *Computer Networks*, Third Edition, 1996, pp 280-281.

[6] USENET post on comp.unix.solaris by Gavin Maltby of Sun Microsystems, 8/22/1997.

[7] RFC 1094: Sun Microsystems, Inc, *NFS: Network File System Protocol specification*, 03/01/1989.

[8] RFC 1813: B. Callaghan, B. Pawlowski, P. Staubach, *NFS Version 3 Protocol Specification*, 06/21/1995.

[9] Personal electronic correspondence with Gregory Neil Shapiro of WPI, 9/2/1997.